What, When, Why and How
Introducing Software Configuration Management in Agile Projects

Authors: Edward Linderoth-Olson, Oskar Präntare

February 28, 2012

Contents

Abstract

This report mainly proved the conclusions of two previous theoretical reports on how to introduce software configuration management in a projec. This is achieved by practically testing said introduction on an actual team. The conclusions answered questions like how software configuration management can be adapted to an agile project, if the composition of the team can affect the introduction of software configuration management and when the various software configuration management activities should be introduced. The main conclusions are the need to adapt the software configuration management to the agile development by introducing for example an iterative feedback and improvement process and how software configuration management can be introduced using iteration zero as a vehicle.

1 Introduction

As Moreira[1] mentions even the most experienced software configuration managers still have over a 33% risk of failure when introducing software configuration management (SCM) in a project. One of the main reasons for failure according to Moreira is the lack of commitment by the administration and/or the developers, something which will be discussed in this report for example as of a consideration of maturity.

As far as we have discovered all research on SCM in agile projects, for example the article written by Bendix[2], deals only with how the different SCM practices can be adapted to an agile environment, and only in passing addresses how to introduce them. We have managed to identify some literature, however it merely tangentially addresses introducing SCM while discussing SCM from a more general perspective.

To solve this problem a theoretical report has been written which presents a hypothesis on what to introduce, when to introduce it and how to introduce it into an agile project. The aim of this report is to discuss the conclusions and through practical experience support or refute this thesis.

The aim of the background chapter is to provide the necessary foundation of knowledge required to understand the other discussions in the report. In the method chapter we discuss how the various topics are included in an eXtreme programming environment. The implementation discusses how the topics from the theoretical report[3] are implemented in this practical project. Results describes the results received from the given implementation in the practical project. Interviews discusses the other team's results and implementations.

[1] Mario Moreira, 'The 3 Software Configuration Management Implementation Levels', *Lecture Notes in Computer Science* 1675 (1999).

[2] Lars Bendix and Torbjörn Ekman, Chap. Software Configuration Management in Agile Development, in: 'Agile Software Development Quality Assurance', (Information Science reference, 2007).

[3] Edward Linderoth Olson Fredrik Karlsson Andreas Nordén, Daniel Perván, 'What, When, Why and How Introducing SCM in Agile Projects', (2011).

2 Background

The purpose of this background chapter is to provide the necessary information for understanding this essay's conclusions. As such it includes an overview of SCM, eXtreme Programming, the course in which the project was undertaken and Kanban.

2.1 Software configuration management

SCM is a method for managing the evolution of a software development project. Bendix described it as "Software configuration management is the discipline of organising, controlling and managing the development and evolution of software systems"[4]. According to the Institute of Electrical and Electronics Engineers the discipline can be subdivided into four different areas of management: Configuration identification, configuration control, configuration status accounting and configuration auditing[5]. Through these four processes SCM handles the software development changes through the entire lifetime of a project[6].

Configuration identification is the management of identification and order of the items (a configuration item is any item within the project which are subjec to the SCM process, for example a configuration item may be a file of code) within the project which are to be managed by the SCM process. For example the selection of items, documentation and naming procedure. In eXtreme programming there is no official practice of how this management should be implemented in the process, however the use of version control forces this implicitly[7].

Configuration control is one of the main areas of traditional SCM, most commonly in the form of a Change Control Board (a Change Control Board is the organ in a software development project which prioritises the implementation of different pieces of functionality, also known as change requsts.), as described by[8]. Regarding eXtreme Programming, the planning game can be viewed as a more informal version of this process, see[9]. However, another important aspect of the change management area of configuration control is not inherent in eXtreme programming, namely traceability between change requests (roughly equivalent to eXtreme programming's stories or tasks) and code.

Configuration audit is the process of assuring that the configuration items conform to the configuration documents (the documented rules of how configuration should be managed), that they are all present and accounted for and that they are of the correct versions. This can be subdivided into two specific types of audit: Physical and functional. Physical audit is the verification that everything is present, and is usually carried

[4] L. Bendix, 'Configuration Management 1:a: Introduction and motivation', *lesson papers of EDAN10* (2011).

[5] IEEE, *IEEE Standard for Software Configuration Management Plans - Redline*, 2005.

[6] Bendix and Ekman (as in n. 2).

[7] Torbjörn Ekman Ulf Asklund, Lars Bendix, 'Software Configuration Management Practices for eXtreme Programming Teams', *Proceedings of the 11th Nordic workshop on Programming and Software Development Tools and Techniques, Turku, Finland* (2004).

[8] M. A. Daniels, Chap. Chapters 2 and 3, in: '', (Advanced Applications Consultants, Inc., 1985).

[9] Ulf Asklund (as in n. 7).

out in preparation for a release to ensure that no configuration items are left out. Functional audit is to ensure that all the ordered functionality has been implemented, that no change requests have been unintentionally left out.

Configuration status accounting is the practice of documenting the status of all configuration items and the changes made to them. In eXtreme programming the status of the configuration items can be seen by keeping track of stories, tasks and their current state[10]. Keeping documentation of the code and keeping a history of the system and its changes is also main activities of the status accounting[11].

2.2 eXtreme programming process

eXtreme programming is a methodology for producing high quality software quickly and efficiently by learning to adapt to changes in programme specifications, the adapting to changes being commonly regarded as being the greatest obstacle to quickly and efficiently producing high quality software[12]. eXtreme programming's major practices are the planning game, small releases, the metaphor, simple design, tests, refactoring, pair programming, continuous integration, collective code ownership, on-site customer, 40-hour weeks, open workspace and just rules. Explanation of each of these is the following[13] .

Planning game The planning game is where programmers estimate the effort required to implement certain functionality, referred to as stories, whilst the customer uses this information to decide what stories should be implemented and when.

Small releases Small releases is the ability to create a release of the system often and quickly without having to wait for the project to be completely finished.

Metaphor The metaphor practice is that the system can be explained through metaphors.

Simple design Simple design is the practice of having minimal amount of code to fulfill the desired functionality.

Test driven development Through test driven development the tests which test an functionality is written before said functionality is introduced.

Refactoring Throughout development the code should be refactored to minimize the amount of code and bad smells while still maintaining all the functionality.

Pair programming Pair programming is when all code should be written by a pair of programmers working together to tackle complex problems.

Continuous integration Continuous integration is the practice of continuously integrating the code the programmers write.

[10] Ulf Asklund (as in n. 7).

[11] Bendix and Ekman (as in n. 2).

[12] Chromatic, *Extreme programming pocket guide*, (O'Reilly, 2003), Pocket References Series ⟨URL: `http://books.google.com/books?id=7SoTzgrOqaoC`⟩, ISBN 9780596004859.

[13] K. Beck, 'Embracing change with extreme programming', *Computer*, vol, 32 oct (1999):10, ISSN 0018–9162.

Collective code ownership Collective code ownership is when all programmers may change any code within the system.

On-site customer The practice of on-site customer is that a customer should always be near the development as to be able to answer upon programmer questions at any time.

40-hour week 40-hour weeks is that no one should work any kind of overtime.

Open workspace Open workspace is that everyone should sit in the same room.

Just rules Just rules is that by being part of an extreme programming team everyone should follow the rules, though that said rules can always be changed by the team.

2.3 EDA260

EDA260 is a programming course taught by the faculty of engineering at Lund University. EDA260 has a focus on practicing the extreme programming methodology within an iterative, agile, process. The course goals are to teach theoretical and practical knowledge of working within a team, with a customer, planning, implementation, testing and all parts of the software development process[14].

The course is divided into a theoretical and a practical part. The practical part is where groups of students are led through the software development, practicing the aforementioned extreme methodology process with the help of older students who lead the course as coaches.[15][16]. This is also where the results of this study stem from.

In EDA260 before the start of an eXtreme programming project an iteration zero is introduced. During this iteration the coaches create a minimal but complete system which has the project's basic architecture in place and creates something that can be discussed and evolved[17]. For a more in-depth discussion of the course please refer to Görel Hedin's article describing the course[18].

2.4 Kanban

Kanban is a method for managing traceability and scheduling of tasks. The Kanban methodology used during this project was the one taught by N. Fors and N. Hansson during the spring 2010. The purpose of Kanban is stated as "visual feedback, self-driven teams, continuous improvement and when functionality is done, [...]"[19]. Kanban was developed at Toyota though it is now more often used in software development.

[14] B. Magnusson G. Hedin, 'Kursprogram Programvaruutveckling i grupp - projekt', (2011).

[15] Ibid.

[16] G. Hedin, 'Kursprogram Coaching av programvaruteam - projekt', (2011).

[17] B. Magnusson G. Hedin, L. Bendix, 'Teaching extreme programming to large groups of students', (2003).

[18] G. Hedin, L. Bendix and B. Magnusson, 'Teaching extreme programming to large groups of students', *Journal of Systems and Software*, 74 (2005):2 ⟨URL: `http://www.dx.doi.org/10.1016/j.jss.2003.09.026`⟩.

[19] N. Hansson N. Fors, 'Kanban i Extreme Programming', (2010).

In Kanban the customer writes down a story on a card, whereupon the effort of implementing the feature is estimated by the programmers and then the feature is given a priority by the customer. The cards are afterward put up on a physical board. Said board was called by N. Fors and N. Hansson ''Kanbanbräde"[20] which in this report will be known as the Kanban board. The Kanban board show which stories that hasn't been started on, which is currently being completed or what stories that has been completed. While the project proceeds the cards will then transfer over the table and show the progress of the project[21].

3 Method

We have based our research on the theoretical studies of two groups in the SCM course EDAN10 at Lunds university of technology. The studies is of how to implement SCM to agile project with a problem formulation similar to above. Our goal is to supplement their theoretical study with practical findings and results, thus either refuting some of their conclusions or supporting them with empirical evidence.

The study will hopefully also answer the following questions, which can be considered our problem domain:

- Do different factors such as group maturity, experience and project development structure influence the introduction of SCM?
- How can SCM follow the agile iterative process?
- How can SCM be adapted to frequent releases?
- How can SCM be adapted to developer feedback?
- What SCM practices are critical in an agile environment?
- In what order should these practices be introduced?

In order to collect data we will use reflections halfway through the project and after it's termination. We have also utilised interviews with the coaches of other teams in order to study the differences in SCM implementation and scope between the projects.

In order to test the implementation of traceability we will add a story whose specific purpose is to be removed in a latter iteration, this will be included in the final version of this report.

Also, the mechanisms for constant improvement (primarily evaluation meetings where everyone in the team discussed and evaluated the latest weeks practices) inherent in the XP methodology have provided us with some additional qualitative information.

The team has followed the plan suggested in the theoretical report, see[22], the report's plan is detailed below together with special considerations for the given project.

[20] N. Fors (as in n. 19).
[21] Ibid.
[22] Andreas Nordén (as in n. 3).

We have successfully implemented all of the required processes and also proper commit comments and some limited traceability support. An in-process audit is carried out to some extent by the coaches, in tandem with assuring that the eXtreme programming practices are observed.

1. Project start

   - Software configuration management documentation

     The coaches decided that the project was too small in scope to merit a full scale up-front configuration management plan, however a discussion with the team was held in order to determine what artifacts were to be deemed configuration items. The result of this discussion was the inclusion of source code and unit tests among the configuration items. Also, the power of designating future configuration items was delegated to the development team resulting in the addition of acceptance tests, acceptance test input files, build scripts, technical documentation and the end user manual. During this meeting a template for commit comments was also drafted and used.

   - Naming conventions

     During the preparatory configuration management meeting naming conventions for all configuration items and coding conventions were decided upon. These were communicated to the development team by means of a preparatory spike and by the implementation of Iteration 0.

2. Release 1

   - Automation of releases

     The automation of the release was done through the implementation of ant-script which would create a finished archive of everything that should be present within the release. Ant is a Java library which in this project was used to automate the team's build process[23].

   - Identifying configuration items

     The identification of new configuration items was performed through a decentralised decision making process were the developers creating the artifacts decided whether these merited inclusion as configuration items. As mentioned previously this resulted in the inclusion of acceptance tests, acceptance test input files, build scripts, technical documentation and the end user manual.

   - Physical Audit

     At the first release a story containing a list of everything that said release was to contain was formulated. An item on the list was to perform a physical audit of the release and test said release before sending it to the customer.

[23] *Apache: Ant-script*, 9/2-2012 ⟨URL: `http://web.archive.org/web/20110722165245/http://ant.apache.org/`⟩.

3. Later in project

   - Story impact analysis

     At each planning game all stories had to be estimated on a scale between 1 and 20. At the second planning game we introduced a general discussion of the impact of a certain story as a means of analysing it during the estimation processs. This discussion was then written down by designated secretaries in the team.

   - Refactoring impact analysis

     Each refactoring was written down as a story which was estimated at the planning game and an impact analysis similar to the one done in story impact analysis was made.

4. Already Implemented

   - Use a version control tool

     The team uses SVN, an open source version control tool with a focus of simplicity of usage[24].

   - Use a build tool

     The team used the Eclipse development environment[25] together with ant-script to automate the build process[26].

   - Incremental Refactoring

     The team should always refactor before implementing a story, after implementing a story and if some larger problems still remain the team should write refactoring stories as to maintain an incremental refactoring where no refactoring grows too large.

   - Keep the Repository Clean

     In iteration zero the tests, code and documentation that should be contained in the repository was defined. The team was given the task to keep the repository clean by following the structure defined in the repository.

5. Optional

   - Write Proper Commit Comments

     During the pre-project SCM meeting a template for writing commit comments was implemented including the authors of the commit, the story implemented by the commit and the reasoning behind the manner of the implementation.

[24] *Apache: Ant-script*, 9/2-2012 ⟨URL: `http://web.archive.org/web/20110725011856/http://subversion.apache.org/`⟩.

[25] *Apache: Ant-script*, 9/2-2012 ⟨URL: `http://web.archive.org/web/20110729024232/http://www.eclipse.org/`⟩.

[26] 'Apache: Ant-script' (as in n. 23).

- In-Process Audit

  At each iteration the coaches continually conduct an in-process audit to maintain the eXtreme programming and SCM practices. Also at each iteration the team writes a summary, a reflection, of how well they consider that they have been able to maintain the practices. At each planning game the team there's a so called evaluation meeting where the team discusses their success at maintaining the practices and what should be done to either keep maintaining the practices or else to include the practices which have failed.

- Traceability

  The SVN version control tool keeps a history of all changes that have been made, the time of said change and by whom it has been committed[27]. This is augmented by a template of proper commit comments defined to make sure that all changes could be tracked to it's author, story and the reason of said change. All stories, analysis, contracts and releases were continuously maintained by designated secretaries within the team. Also, a Kanban board was used to track the progress of stories and the team's current occupation.

- Simplified Branching Strategy

  Originally this was intended to be studied but due to time and other project contraints it had to be excluded from the report.

4 Results

In the results chapter the results of each of the parts mentioned in the implementation chapter and interviews with other teams are discussed. This is the report results of the SCM implementation.

4.1 Software configuration management documentation

The pre-development stage discussions of configuration items and naming conventions seem to have served the team well, providing the rudimentary groundwork which the developers have then continued to expand and improve on. In this latter respect the delegation of decision making power to the developers has served the team well in including all necessary files whilst avoiding unnecessary clutter of the repository. This also rhymes well with the agile philosophy and the new role of configuration management managers in agile projects as outlined by Bendix[28]. Throughout the questions appendix the team's answers point towards the current documentation being sufficient, all comments of question 2 said that the current documentation has been enough. Considering the team's answers to the questions 11 and 12 it seems that they prefer asking people questions rather than reading the documentation that currently exists.

[27] 'Apache: Ant-script' (as in n. 24).
[28] Bendix and Ekman (as in n. 2).

4.2 Naming Conventions

The naming conventions put down during the initial discussions have, as previously mentioned, served the project well and facilitated the identification of configuration items using natural language. As the project is very limited in scope there has been no problems in using this convention due to the number of configuration items.

4.3 Automation of Releases

The automation of releases made the team able to create a release within a few seconds. As the stories included in the release were often completed only a few minutes within the deadline the automated release was very welcome as otherwise the building of the release could have caused large delays. According to the answers to question 4 the team considered that the automation has helped them throughout the project.

4.4 Identifying Configuration Items

The identification of configuration items has worked well, the necessary expansions have been made as the project has grown while the clutter in the repository has been kept to a minimum. The answers to question 1 indicated that the team considered that the repository was kept somewhat clean during the project.

4.5 Physical Audit

Team 4's releases were always runnable on the customer's system and contained everything said releases were supposed to contain, whilst taking a very small amount of time to build. The automation of the physical audit using an ant-script has proven particularly useful, and the rewards of implementing this as early as possible in the project cannot be stressed enough. The answers to question 5 "What do you believe has been the main reason of why team04's releases never have had any files missing which the customer has asked for?" were for example that the release was tested, that the ant-script generated all content, communication and the coaches reminding them of everything that should exist within a release.

4.6 Story Impact Analysis

The story impact analysis was discussed and noted. Throughout the first iterations there were tendencies among the team to decide that the story impact analysis must be wrong and that the story's points had to be decimated. This did however prove a dire mistake and led to the delay of release 1B, but the team seems to have learned from this mistake. A further discussion of this topic will have to be delayed pending further data from the project. The answers to question 6 points towards that the team considers that the effort estimation has been hard but they are getting better at it.

4.7 Refactoring Impact Analysis

The same as story impact analysis. Even if the refactoring impact analysis was discussed and documented it was still disregarded when making decisions. The answers to question 7 states that it is a lot harder than story analysis, according to them mainly as they haven't experienced any major refactoring in any earlier projects.

4.8 Write Proper Commit Comments

Commits was always properly commented, providing rudimentary traceability and the ability to find when errors were introduced or who to ask about a specific piece of functionality. This could be done as to see when what code was commited, whom the authors were and similar information which could be collected through the use of SVN or reading the commit comments, specified in the implementation chapter. Though because of the ease of instead calling the team to stand-up meetings the team, according to the answers to the questions in the appendix, haven't used the commit comments to any larger extent.

4.9 In-Process Audit

The continuous in-process audits by the coaches has made sure that the team in general maintained enough of the practices to at least avoid having the coaches comment of their malpractices. When the team gets too relaxed with any practices and stop using any them often it shows in the beforementioned reflections at the end of the day and the team comes up with a solution of how to avoid said problem in the future at the planning game. No practice has been forgotten for more than one iteration at a time. According to the answers to the question 9 in the appendix the team agrees that the in-process audits by the coaches and the reflections are the main components motivating them to follow the eXtreme programming practices.

4.10 Traceability

The version control tool's inherent traceability together with commit comments made the team able to find when and where errors were introduced or whom to ask about specific functionality. See the proper commit comments subsection for a more specific explanation. The Kanban made the team able to see which stories were worked upon and by whom. Even if at the start of the project each card with a story was designated to a pair of programmers the pair switches made it difficult to keep the position of the developers updated. After switching the card's connection to the workplace where the card is worked upon rather than the pair which is working on it the card's progress was better maintained. When the cards weren't updated the team had problems finding who were currently working on a specific story. Cards that did not have a multiple of it to be used by the team at their workplace started to be picked down from the board and thereby creating a loss of traceability. The moment this happened people stopped noticing who were working on said stories, as they were removed from the board, causing many pairs to work on the same story. To fix this copies were created of

all cards that didn't have any multiple of it when they were received by the team. This made sure there always existed a card to be brought to the workplace by the team while also having all stories traced on the Kanban board. According to the team's answers in the appendix there exists traceability, especially through the Kanban board, though traceability which has to be reached through the version control tool wasn't used but instead asking the team at stand-up meetings was preferred.

4.11 Simplified Branching Strategy

As the extent of the project was very limited there was, as previously mentioned, no opportunity for investigating branching.

5 Group software configuration management maturity

During the first planning meeting the development team filled out a questionnaire asking for solutions to several common problems related to SCM. The purpose of this questionnaire was to determine the teams SCM maturity, how well-versed they were in its processes, advantages and pitfalls. In general a large disparity was noted with some members displaying some previous knowledge and others none. Overall the team was aware of how to make use of SCM processes to support concurrent development but ignorant in other areas, and for all intents and purposes the team could be regarded as having no background information at all. The team considers the group's maturity to affect the project, sometimes positively and sometimes negative, though the negative problems could usually be solved according to the team's answer to question 4 in the appendix by spikes or communication.

6 Interviews

Depending on the groups' progress in the project the interviews aimed to identify different factors. For groups that had been less productive or experienced problems collaboration the aim was to identify these problems and possible causes. On the contrary the aim of the interviews with the groups that had performed well was to determine why they hadn't experienced these problems, and to identify whether this was connected to SCM practices utilised by the group.

A first attempt at producing a battery of questions was made, however this was concluded to be of too general a nature to actually be of any value, and so instead an approach of holding general reflective discussions within a few predetermined topics was adopted.

6.0.1 Team 03

There had been problems with the first release, primarily caused by incomplete functional audit (testing) of some extra stories. All the promised stories made it into the release intact. In order to assure a complete functional and physical audit the team wrote an ant-script which ran all tests and if these passed built a release.

The identification of configuration items has worked well, there have been no incidents of files being missed out. On the other hand, there was some caution on the part of the coaches as to whether the team might be somewhat too liberal in their view of what was a configuration item, leading to some minor cluttering of the repository. This was however regarded to be limited in scope and cleaned out as the project progressed, allowing it to be kept within permissible levels.

6.0.2 Team 05

The initial release of the system had many problems, for example the programme crashing and the manual not being included. No audit, neither functional nor physical, was performed for this release resulting in, among other things, the manual template being sent instead of the manual itself. In preparation for the complementary release 1B audits had been implemented in the form of a shell script and the release was without problems.

Lack of communication led to collaboration problems between developers and as a result duplicated work and double maintenance issues. The coaches were of the opinion that this was most likely a result of difficulties in creating a team spirit rather than any problems in the SCM process.

A very well-developed and utilised system for commit comments has been used by the team, every comment including the authors, version of the class, time of commit, story, affected methods and a short description of the modifications. Also, the java documentation includes a specified author for every method which is last person to modify it. The coaches expressed some reservations as to the usefulness of all this extra documentation the production of which apparently was consuming a large amount time, however emphasised that the team were strongly in favour of it and regarded it as a great help in their work.

The team had to a large extent utilised branches to facilitate large scale refactorings, with mostly good results. There were some initial problems due to the team being unfamiliar with SVN's merge process, however a training spike with practical experiments solved this and eliminated the problems.

6.0.3 Team 06

Due to a scheduling error this group, who had not experienced too many problemsl, was interviewed first and so the possible errors they had avoided was based only on the experience of Team 04. As such the discussion took a slightly more general turn.

In the coaches' opinion team 06 was very independent requiring very little involvement on their part and overall performing very well. The group had yet to implement any manner of formalised physical audit, however they did spend considerable time preparing each release so it can be presumed it was performed only not documented. Pair switches are performed at the end of tasks, which should help preserve continuity.

There have been problems with code disappearing, however the team has been unable to determine the cause and have put it down to a SVN malfunction.

The most notable aspect of the group composition is that it has an overall high level

of programming experience, although some of the members are somewhat unenthusiastic regarding the XP methodology.

6.0.4 Team 10

The teams releases had been automated from the very beginning of the project, and they had never experienced any items being omitted or incomplete.

There had been some issues with the identification of configuration items, customised input files were used for many of the tests but were not included in the versioning system, causing the tests to fail when not run on the computer they were written on. This was solved by implementing stricter guidelines for how tests were written and removing the external input files.

The group had used Kanban for synchronisation and tracking of tasks throughout the entire project.

The coaches were of the general impression that the team was very independent and self-sufficient, requiring very little intervention on their part.

7 Discussion

As described SCM can be adapted to an agile iterative process without any problems. Even if they are not explicitly stated in the agile methodologies said methodologies often implement SCM unofficially through the tools they use during development. In traditional SCM managing and controlling changes takes a lot of effort and time from development, in contrast to the agile concept of to welcoming change. A solution is to make the management more oral and try to automate as much as possible of a change. SCM can be adapted to frequent releases by automating the release processes, allowing the team to create a release within a few seconds thus adapting SCM to frequent releases must be seen as a success. Which is the answer to the same question in the problem domain.

Early traceability through version control tools and proper commit comments has been of good use throughout the project. Kanban was also of good use, proven as the moment something caused the Kanban to not be updated wider problems throughout the project occurred. It transpired that the team required that the Kanban board be easily updated. If the Kanban board wasn't easily updated then the maintenance of the board would be forgotten and thereby making it hard for the team to trace who was working on what story. Also if the stories which were worked on weren't visible seen on the table the pairs were prone to start working on the same stories.

Physical audits were as mentioned introduced before the first release and as that the team has yet to fail to deliver required documentation or files at a release we have considered it a success.

The largest problems during the project has been to make the team read code which they themselves haven't written and to make precise effort estimations. Even if all SCM practices were in place they were disregarded in these areas. Therefore even if coaches can implement SCM practices to avoid problems it is of no use if they are not followed. So the group maturity, experience and project development structure did

influence the introduction of SCM, for example by disregarding said introduction. We expect however that as the in-process audit has been able to keep most of the practices in use this will mainly be a problem during the introduction of SCM, though due to the short duration of the project this will be excruciatingly difficult to verify. So answering the problem domain question ”Do different factors such as group maturity, experience and project development structure influence the introduction of software configuration management?”, yes, it does.

As the introduction of SCM into the team was successful we consider that the problem domain questions of criticality and of which order to implement SCM practices in a system was answered to be the same as the given answer in the implementation chapter.

An important aspect of introducing SCM into an agile project seems to connect this to the iterative feeback and improvement process, in order to discover areas for improvement and possibilities for reducing overhead. In this project this can for example be seen as how the team suggested introducing mandatory copies of each story located on the Kanban board. Which is a summary of the report topic of how SCM can follow the agile iterative process and how to adapt to developer feedback.

8 Conclusion

In general the SCM introduction could be seen as a success and that the importance of trying to follow the process which it was introduced into was seen. The main important parts of the SCM scheduled for inclusion at the beginning of the project should be considered iteration zero components, closely followed by the first release components. Then at each iteration the SCM evolves together with the team, based on feedback and seen defects it grows either more or less complex and adapts to the needs of the team and the customer.

9 Acknowledgements

This report could not have been written without the cooperation of everyone throughout the course. We, the authors, wish to thank the coaches of the teams 3, 5, 6 and 10 for their help with answering our questions, the teachers whom are involved in the course as they allowed all of our crazy ideas and our neighbouring team, team 1, as of not going nuts from our loud discussions. Though foremost and most we have to thank our own team, team 4, for the good times we’ve had and that without them this report would be pointless. Thanks!

References

Apache: Ant-script, 9/2-2012 ⟨URL: http://web.archive.org/web/20110722165245/http://ant.apache.org/⟩.

Apache: Ant-script, 9/2-2012 ⟨URL: http://web.archive.org/web/20110725011856/http://subversion.apache.org/⟩.

Apache: Ant-script, 9/2-2012 ⟨URL: http://web.archive.org/web/20110729024232/http://www.eclipse.org/⟩.

Andreas Nordén, Daniel Perván, Edward Linderoth Olson Fredrik Karlsson, 'What, When, Why and How Introducing SCM in Agile Projects', (2011), Student project in Configuration Management course at the Faculty of Engineering, Lund University.

Beck, K., 'Embracing change with extreme programming', *Computer*, vol, 32 oct (1999):10, pp. 70 –77, ISSN 0018–9162.

Bendix, L., 'Configuration Management 1:a: Introduction and motivation', *lesson papers of EDAN10* (2011).

Bendix, Lars and Ekman, Torbjörn, Chap. Software Configuration Management in Agile Development, in: 'Agile Software Development Quality Assurance', (Information Science reference, 2007).

Chromatic, *Extreme programming pocket guide*, (O'Reilly, 2003), Pocket References Series ⟨URL: http://books.google.com/books?id=7SoTzgrOqaoC⟩, ISBN 9780596004859.

Daniels, M. A., Chap. Chapters 2 and 3, in: '', (Advanced Applications Consultants, Inc., 1985).

G. Hedin, B. Magnusson, 'Kursprogram Programvaruutveckling i grupp - projekt', (2011).

G. Hedin, L. Bendix, B. Magnusson, 'Teaching extreme programming to large groups of students', (2003), pp. 136, 139, 140.

Hedin, G., 'Kursprogram Coaching av programvaruteam - projekt', (2011).

Hedin, G., Bendix, L. and Magnusson, B., 'Teaching extreme programming to large groups of students', *Journal of Systems and Software*, 74 (2005):2, pp. 133–146 ⟨URL: http://www.dx.doi.org/10.1016/j.jss.2003.09.026⟩.

IEEE, *IEEE Standard for Software Configuration Management Plans - Redline*, 2005.

Moreira, Mario, 'The 3 Software Configuration Management Implementation Levels', *Lecture Notes in Computer Science* 1675 (1999).

N. Fors, N. Hansson, 'Kanban i Extreme Programming', (2010).

Ulf Asklund, Lars Bendix, Torbjörn Ekman, 'Software Configuration Management Practices for eXtreme Programming Teams', *Proceedings of the 11th Nordic workshop on Programming and Software Development Tools and Techniques, Turku, Finland* (2004).

A Questions 13/2-2012

The team was asked a few questions of software configuration management as to answer how the introduction of software configuration management had worked according to the team. The team could answer the questions either as individuals or as pairs depending on their own wishes. Below each numbered answer represents a group's answer for said question.

Has the repository felt clean or has it been filled with garbage during this course?

Answer 1

Most of the time it has been clean. Several times junkfiles has been commited but they were due to team feedback removed by recommiting with a cleaned recommit.

Answer 2

Yes, it's been really smooth, we've only encountered garbage once during these iterations.

Answer 3

The repository hasn't been filled with garbage, although it wasn't clean all the way.

Answer 4

The repository has generally felt very clean. There has seldom been any red code or irrelevant files in the current revisions. There has been a few times that somebody forgot to check their unit tests before committing, but this has been a rare occurrence.

Answer 5

I think the repository has been clean, not a lot of out-commented code or anything.

Answer 6

Sometimes there has been lots of garbage files, but usually it has been pretty clean.

Has the documentation of the code during the course been sufficient for you to understand what the team has created?

Answer 1

Yes, most of the code has been properly commented during the whole time, except for the test-classes.

Answer 2

Yes, sometimes it's been even more then sufficient, border to overflow.. Its been abit to much to maintain to benefit from the help it gives.

Answer 3

Sufficiently

Answer 4

The documentation has often been sufficient enough to help one understand the code. But there has been several times where there has been undocumented code. In these case one could often figure it out what the code did either way. Test cases could have been documented more thoroughly.

Answer 5

Yes, i rarely read comments, I just look at method names/code.

Answer 6

Yes, we think so.

Has it been hard to agree on naming conventions for the files that have been under version control?

Answer 1

No, we have agreed to use Java naming convensions for code and classfiles. The rest of the files has used common names that is often used in this types of projects.

Answer 2

Overall it's been easy to agree with both the naming conventions of java and the team. But somethings like "Endtime", "Goaltime","Stoptime" has been annoying.

Answer 3

No it hasn't.

Answer 4

It hasn't been hard to agree, but renaming classes has been hard and people have been reluctant to do it due to SVN bugging out.

Answer 5

No, the names were decided during meetings where everybody was part of naming them.

Answer 6

No, but it has sometimes been hard to find a good name for files.

What do you think of automated releases with an ant-script?

Answer 1

Has has worked very nice. ANT-scripts are easy to learn and use. You have to be careful when refractoring, pakage- and class-renamning can break the script.

Answer 2

Since our ant-scripts first impression to us was really buggy you always felt something in your guts when you used it, but with time it eveloped and got really beneficial to use.

Answer 3

It seems effective, although we haven't used it by ourselves.

Answer 4

The Ant-script has saved us a lot of time when releasing to the customer, as long as we remebered to update it if we change the name of the classes.

Answer 5

I have not been very involved in that process but to me it seems that it is semi-smooth. Sometimese there is a problem with yhe script but I guess it beats compiling and gathering files by hand.

Answer 6

Convenient.

What do you believe has been the main reason of why team04's releases never have had any files missing which the customer has asked for?

Answer 1

Mainly due to the ANT-script buildning the dists.

Answer 2

The main reason, besides our coaches mindblowing expertise, have been our communication, and with that support the communication lead to. With that we mean not only the communication has been a part of the sucsess but the support you received when you communicated, for example if you encountered a problem you never seen before there where always somebody there to help you.

Answer 3

The release was put through several tests before it was sent to the customer.

Answer 4

We have had good customer contact, as well as our coaches has reminded us of the release requirements.

Answer 5

A well oiled system for which tasks to implement and their priority.

Answer 6

We think the people responsible for the release have been thorough.

How well have you been able to effort estimate stories?

Answer 1

We had various opinitions, but easily agreed on your estimations during your meetings.

Answer 2

Not that well, each week we tried to hard getting ”yesterdays weather” right and manipulated the scores so that they never got to adjust them self. For example one story we cut the score in half just since it felt ”to much” but whilst implementing it we noticed that it actually was worth its higher score.

Answer 3

It's improving, it was worse in the beginning.

Answer 4

It's been hard to give an exact estimate, some are undervalued and some are overvalued. But in average, we have completed as many points as we set out during the iterations.

Answer 5

In the beginning of the course I often underrestmated the omplexity pf the stories but now, later on, it gets more and more precise. I still tend to underrestimate but it is less extreme now:

Answer 6

It has been a bit rocky, but we think it's stabilzing. We think that planning poker was a good idea.

How well have you been able to effort estimate refactoring?

Answer 1

We had various opinitions, but easily agreed on your estimations during your meetings.

Answer 2

We don't dare to talk for all of our in our team but since we two haven't really refactored any major project before it's been really hard to estimate the effort going into a major refactor.

Answer 3

Not that well. The refactoring usually turns out to be harder than estimated.

Answer 4

We have often underestimated the value of refactoring stories, mostly due to not realizing how much the dependancies and tests are effected.

Answer 5

Better that story estimating. My estimations have been quite good for refactorisation.

Answer 6

This has been very hard. It's not easy to estimate.

If there has been any failure of following the agreed amount of effort for a iteration, what has according to you been the main reason?

Answer 1

During planning of the first iteration we overestimated what we planned to accomplish. Under the second we underestimated the effort, which resulted in 4X accomplished points.

Answer 2

The only problem have been the same as on the question ”How well have you been able to effort estimate stories?”.

Answer 3

Wrong estimation of stories and tasks. Pride?

Answer 4

Most often it came down to that we underestimated a few stories. However, we have been able to finish them during the next iteration.

Answer 5

Underestmation

Answer 6

Sometimes, we think we know more or work faster than we actually do, or underestimate the work necessary to complete a task.

If you have been following the eXtreme programming methodology, what has been the main reason why you have been able to follow it or avoided forgetting to follow it?

Answer 1

Our coaches has strongly encuraged XP methodology. Communication between navigator and driver has also enforced this metodology.

Answer 2

Neither of us have been in a big project as this one, so the atmosphere of this project and team has been keeping us to follow the XP methodology. But also the fact that you have have a ”focus goal” each week has help to learn how each part of the methodology works.

Answer 3

It's easy to understand, works very well, and our coaches are constantly reminding us about the rules.

Answer 4

Pair-programming has been a lot of help when using XP. One takes fewer shortcuts, update the tests more often, and (almost) never commits bad code.

Answer 5

I have problems to remember and motivate myself to write the tests before I white the code.

Answer 6

We think that pair programming has enabled us to follow XP, and the hard pushing of TDD.

Have you been able to find who is working on what story? If so, how? If not, do you have any suggestions as to how this could be fixed?

Answer 1

Kamba with whiteboards has been used. This has been working pretty well.

Answer 2

Yes, much thanks to the Kanban system.

Answer 3

Yes, either by asking, or using the letter-system that our coaches have invented.

Answer 4

Kanban has been a very good tool for keeping track of who is working with who on what. Sometimes people have forgotten to update it, but it never takes long to find out.

Answer 5

I have not had any problems with that

Answer 6

Yes, we have had instances where it's hard to know who is working on what. The Kanban should be followed more strictly to fix this.

Have you been able to find who wrote a specific story? If so, how? If not, do you have any suggestions as to how this could be fixed?

Answer 1

Instant standing meetings has helped to figure this out.

Answer 2

The only stories, not written by the customer was almost exclusively written by the entire team, together.

Answer 3

The same way as mentioned in the previous answer.

Answer 4

Finding out who implemented what was done by checking the SVN history, or just calling a meeting to ask.

Answer 5

I have never wanted to know who wrote a story, the stories have been quite good so I have never had the need. to make it easy to find out who wrote a story you could have a creator-tag on each story.

Answer 6

Yes, but only by asking around. Maybe a log of who was working on what story and when should be kept?

Have you been able to find who implemented a specific functionality? If so, how? If not, do you have any suggestions as to how this could be fixed?

Answer 1

Instant standing meetings has helped to figure this out.

Answer 2

No, even though SVN keeps history of commits and we use good commitlogs, people often change pairs, etc, before commits so the credit often don't go to the one who understands the implemented specific functionality.

Answer 3

Collective meeting and reading the repository- history.

Answer 4

[No answer, author's comment]

Answer 5

Not really, group meetings does it for us.

Answer 6

Yes, but only by asking around. See above? Functionality is basically always a story.

Have you been able to find what stories that no one has started working on yet? If so, how? If not, do you have any suggestions as to how this could be fixed?

Answer 1

This is solved by kamba and the "TODO" section.

Answer 2

Yes, thanks to the Kanban system.

Answer 3

Yes, by looking at the ToDo- list on the whiteboard.

Answer 4

Kanban again, has helped us to keep track of unimplemented stories.

Answer 5

Yes, they have all been under TO-DO

Answer 6

Yes, by looking at the section of the Kanban labelled ”TODO”. This is usually correct.

How do you think the group's knowledge has affected the outcome of the various activities above? Has the group's ability to follow agile practices made it easier or harder to perform the various activities above? Why?

Answer 1

Most of us where new to this types of development, but due to spikes in various topic your knowlege has evolved. The ambition to follow XP has been pretty high on average.

Answer 2

We have had the honour to work side by side with very competent people, so yes the groups expertise have certainly effected the outcome, but then again, this project isn't hold back by skill rather than the task to cooperate with the entire team, so the agile practices has made it easier and also effected the outcome.

Answer 3

Everybody had decent and sufficient knowledge about the process, and those who lacked certain fragments of it could easily compensate them by asking other members. Which made it easy to follow the agile practices.

Answer 4

The groups knowledge has been varied, which has helped us sometimes (such as some people being good at Ant Script), but sometimes it has led to confusion and complicated solutions.

Answer 5

Obviously, knowledge make it easier to develop programs, and it has shown over time that when our groups knowledge about the process has increased, so has the pace of development.

Answer 6

It has definitely affected the work, and we think it's obvious why. However, it's much harder to know how it has affected the work, and we don't think we can estimate this.

B Introductory survey

Ifall ett par har en del arbete kvar innan release medan resten av teamet behöver börja arbeta på nästa release, hur kan resten av teamet via SVN kunna arbeta vidare utan att påverka den första releasen som ett par arbetade vidare med?

Person 1

Branching och hålla igång det paralellt tills alla är i fas.

Person 2

Branching, optimistisk eller pessimistisk

Person 3

Ta rast så att de är pigga sen, eller hjälpa teamet som har problem. Börja på kod i nästa release som inte tangerar det teamets arbete.

Person 4

Branches

Person 5

Branches

Person 6

Branch?

Person 7

Brancha

Person 8

Branch

Person 9

Arbeta på avskilda repositories.

Person 10

Använda två repositories

Det största problemet som kan ske vid parallellt arbete med samma kod är merge-conflicts. Desto större skillnaden är mellan det man har i sitt workspace och det som finns i repositoriet desto större blir merge-konflikten. Vilka två metoder kan ni använda för att minimera problemen varje merge-conflict skapar?

Person 1

Inte hamna efter i updates (update ofta). Commit så fort du har fungerande funktionalitet så inte andra som kör update får för stora problem.

Person 2

Undvik onödiga ändringar (whitespace, omformatering, etc.). Uppdatera sin lokala version ofta. Committa så snabbt man har fungerande tester.

Person 3

- Uppdate, uppdate, commit - för att slippa ”big bang”.
- Eftersom vi kör ”kanban” så kan man titta vilka andra som jobbar med ens filer och kanske förklara när man implementerar vissa saker.

Person 4

Committa och uppdatera ofta, samt dela upp arbetet

Person 5

Update och synchronize.

Person 6

Update, update, commit

Person 7

Uppdatera ofta

Person 8

Uppdatera, uppdatera, uppdatera. Synkronisering.

Person 9

Uppdatera koden och testa den. Kommunicera med andra.

Person 10

1. Sammarbeta med det paret som man fått merge-konflikt med.
2. Uppdatera ofta för att slippa merge-konflikter.

Ett stort brott inom XP är att commita in röd kod eller röda tester till repositoriet. Hur undviker du att detta sker?

Person 1

1. Se till att tester fungerar.
2. Uppdate få senaste version.
3. Sen kolla alla tester igen.

Person 2

Script som kontrollerar det, disciplin.

Person 3

Testar innan commit och är säker på att koden klarar alla test... uppdate, uppdate, test, commit

Person 4

Genom att testa all kod innan.

Person 5

Prata med gruppen, se om någon kan hjälpa, committa inte förrän det är löst.

Person 6

Kör tester innan commit.

Person 7

Don't

Person 8

Test innan commit

Person 9

Testa tills det blir grönt, uppdatera, testa igen, uppdatera, testa, commita.

Person 10

Köra alla tester innan commit.

Hur tycker du att teamet ska bete sig för att se mellan varje vecka vad som har gjorts, varför detta har gjorts och vilka stories som detta uppfyllde?

Person 1

Gå igenom det i grupp, någon kort reflektion.

Person 2

Issues i Trac

Person 3

Diskutera reflektionerna som skrevs samt snabbt diskutera hur gruppen tyckte att det gick...

Inte prompt räkna godisbjörnarna!

Person 4

Varje person skriver vad och varför man gjort vad man gjort.

Person 5

Möten, Kanban, Trac, ren kommunikation

Person 6

[No answer, author's comment]

Person 7

Komunisera

Person 8

Möten

Person 9

Jag skulle ta upp det dels på planeringsmöten och dels under måndagarnas händelser.

Person 10

Kolla igenom vad som gjorts hela gruppen och diskutera.

Om man vid skrivandet av manualen använder en kombination av googledocs och latex, och sen upptäcker att man vid kompilering har rättat fel i filen lokalt men inte i googledocen, hur skulle man enkelt kunna lösa detta i ett PVG-projekt?

Person 1

Genom CVS, SVN eller något liknande för att plocka fram ”rätt” version.

Person 2

Lägga in det i googledocs. Undvik att detta händer från början.
Att använda en combo av google docs och latex känns onödigt. Jag vill hellre att vi använder antingen eller.

Person 3

Rätta filen rätt innan man ”committar” till googledocs...

Person 4

SVN

Person 5

Använd SVN istället

Person 6

[No answer, author’s comment]

Person 7

Anv SVN

Person 8

SVN

Person 9

Uppdatera från googledocen och börja om, eller rätta till filen lokalt tills det kompilerar utan fel, och passerar testerna.

Person 10

Lägga in manualen i eclipse-projektet och committa med SVN.